FOUNDATION 5

Foundation est une technologie dérivée de bootstrap permettant de développer facilement des applications web responsive qui ne sert qu'à la partie front-end.
Cette technologie est très souple et très utile pour faire des applications adaptées à toutes les interfaces.

Get started

Pour utiliser cette technologie, il suffit de se rendre sur le site officiel de Foundation (http://foundation.zurb.com/) et de télécharger le dossier de base mis à disposition qui sert de départ à tout projet avec cette technologie.
Il faut veiller à reprendre tous les include concernant Foundation que l'on trouve sur la page de départ lorsque l'on créé une nouvelle page.

Mise en forme avec des grilles

Grâce à un système de grille personnalisable à volonté, similaire à celui de bootstrap et facile à prendre en main, il est facile de développer des sites s'adaptant à différents formats d'appareils.

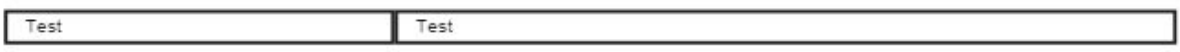
Une grille est définit par une balise div avec la classe row et est découpé par défaut en 12 colonnes, on peut placer une grille n'importe où, y compris à l'intérieur d'une colonne d'une autre grille.

Le découpage de la grille se réalise en définissant des zones avec des balises div dont on peut définir la largeur en nombre de colonnes qui varie en fonction de la taille de l'interface ; foundation distingue 3 formats d'interfaces en fonction de la largeur de celle-ci : small (mobile), medium (tablette), large (ordinateur).
Les éléments se place dans une row de gauche à droite, quand les 12 colonnes d'une ligne sont remplies, les éléments suivants se placent sur la (ou les) lignes suivantes.
Il faut noter que par défaut les éléments en interface small ont une largeur de 12 colonnes si celle-ci n'est pas redéfinie.

Exemple :

```
<div class="row">
       <div class="medium-6 large-4 columns" style="border:solid">Test</div>
       <div class="medium-6 large-8 columns" style="border:solid">Test</div>
</div>
```

Résultat en large :

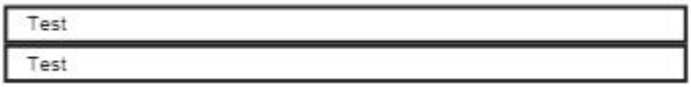


On a bien une zone large de 4 colonnes puis une de 8.

Résultat en medium :

On a bien 2 zones larges de 6 colonnes.

Résultat en small :



On a bien 2 zones larges de 12 colonnes qui se mettent l'une en dessous de l'autre.

Ce système de grille très souple permettant de découper la page à volonté tout en proposant un système simple pour l'adapter en fonction de l'interface est un atout pour faire du développement de manière responsive.

Plusieurs propriétés sont disponibles pour personnaliser davantage le positionnement des zones (offset, centered, float) mais, ayant plus de rapport avec le design que l'adaptation, elles ne sont pas traitées dans ce tutoriel.

Mise en forme avec des blocks

Un autre moyen de mettre en place les éléments dans une page est d'utiliser le système de block de foundation, il offre moins de personnalisation que le système de grille mais permet de mettre en place rapidement et facilement l'affichage d'une liste d'élément en colonnes avec de l'adaptation en fonction de l'interface.

Un block est en fait un élément ul dont on personnalise le nombre de colonne en fonction de l'interface avec la class de manière similaire à la grille :

```
<ul class="small-block-grid-1 medium-block-grid-3 large-block-grid-4">
       <li><img src="imgPosts/yeti.png"></li>
        ...
       <li><img src="imgPosts/yeti.png"></li>
</ul>
```

Toutes les colonnes (pour une interface donnée) font la même largeur, les éléments sont redimensionnés et placés de gauche à droite, de haut en bas.

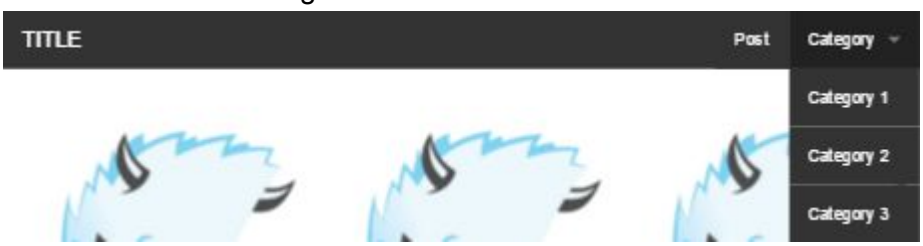
Bien qu'offrant moins de personnalisation que la grille, ce système permet de positionner rapidement et efficacement des éléments de manière responsive.

Menu de navigation

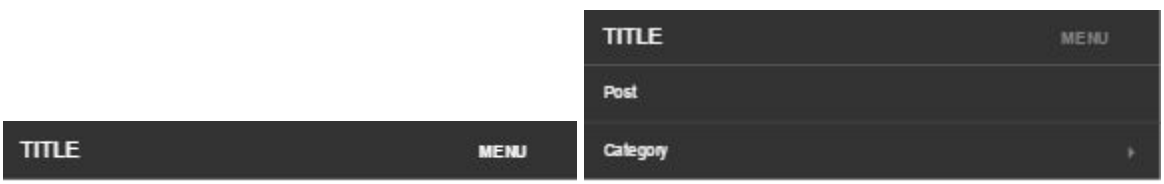
Foundation inclut un système simple pour définir un menu de navigation en haut de la page qui s'adapte bien au format small (mobile), en voici un exemple avec le résultat puis les explications :

```
<nav class="top-bar" data-topbar="">
   <ul class="title-area">
      <li class="name"><h1><a href="index.html">TITLE</a></h1></li>
      <li class="toggle-topbar"><a href="#">Menu</a></li>
   </ul>
   <section class="top-bar-section">
      <ul class="right">
         <li><a href="post.html">Post</a></li>
         <li class="divider"></li>
         <li class="has-dropdown not-click">
            <a>Category</a>
            <ul class="dropdown">
               <li><a>Category 1</a></li>
               <li><a>Category 2</a></li>
               <li><a>Category 3</a></li>
            </ul>
         </li>
      </ul>
   </section>
</nav>
```

Résultat en medium/large :



Résultat en small :



Le tout est englobé dans une balise nav avec la classe top-bar qui délimite ce que foundation va utiliser pour le menu.

La partie title-area permet d'afficher un titre (si on le souhaite) et de définir le bouton qui va afficher le reste du menu lorsque l'on passe en format small (toggle-topbar) ; en effet, tous les éléments suivants du menu sont cachés pour le format mobile jusqu'à interaction avec le bouton menu, et ce dernier est caché pour les formats medium et large.

La partie section contient la liste des différents choix du menu que l'on choisit d'aligner à droite ou à gauche, on peut y placer de simples boutons, des menus déroulants, des saisies de textes, etc …

Comme vous pouvez le constater sur l'exemple, placer des menus déroulants est très simple :
la classe has-dropdown signale que c'est un menu déroulant puis la liste avec la classe dropdown à l'intérieur contient les éléments de celui-ci. La classe not-click permet de préciser que le menu doit se dérouler en passant le curseur dessus, sans avoir besoin de cliquer.

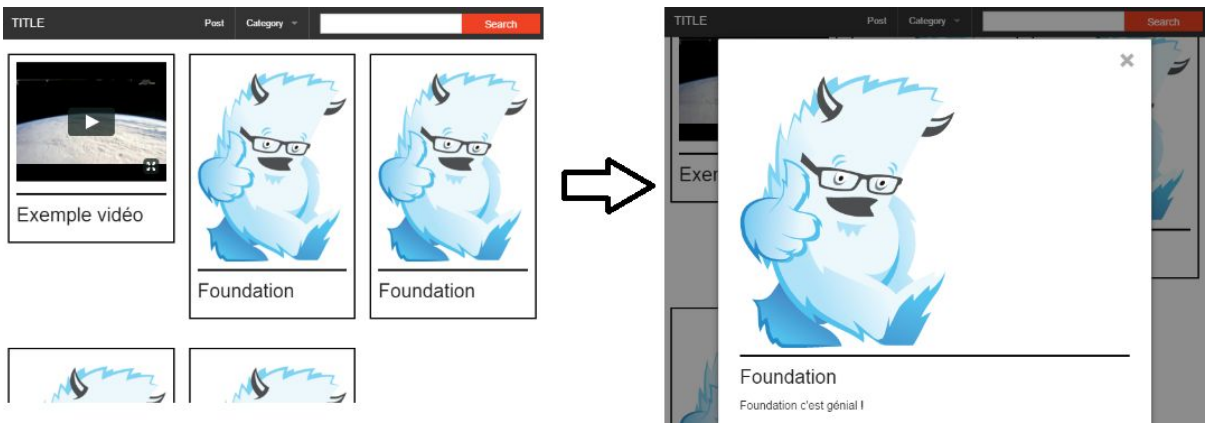
On peut placer des éléments divider entre les différents choix du menu pour une séparation plus visible (peu visible sur le résultat de l'exemple étant donné que le menu category est déroulé).

On peut également englober le tout dans un div ayant pour classe fixed afin que le menu reste disponible sur l'affiche même quand l'utilisateur scroll vers le bas.

Le système de menu de navigation proposé par foundation est donc très facile à mettre en place et s'adapte très bien au format mobile.

Intégration de modal

Cette technologie permet d'intégrer facilement des modals dans vos pages web : quand l'utilisateur clique sur un lien, une image ou autre, une "pop-up" s'ouvre révélant plus de contenu :




```
<a href="#" data-reveal-id="foundationModal">
   <div class="boardPost">
      <img src="imgPosts/yeti.png">
```

```
      <h3>Foundation</h3>
   </div>
</a>
<div  id="foundationModal"  class="reveal-modal  large-10  small-10  medium-10  columns"
data-reveal>
   <img src="imgPosts/yeti.png">
   <h3>Foundation</h3>
   <p>Foundation c'est génial !</p>
   <a class="close-reveal-modal">&#215;</a>
</div>
```

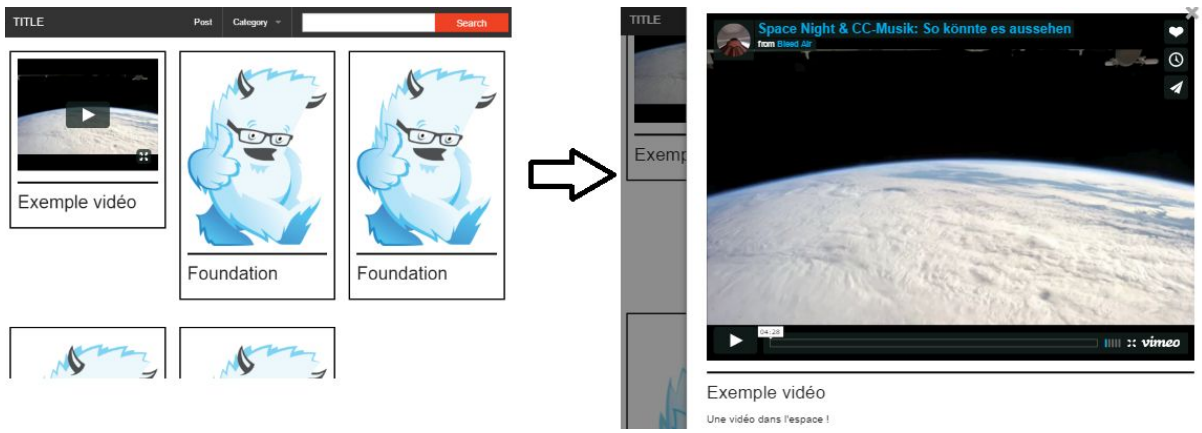
Ceci se réalise en rajoutant une balise a avec un attribut data-reveal-id qui va, lors d'un clic, déclencher l'affichage du contenu caché ayant l'id indiqué dans l'attribut précédent, et ayant la classe reveal-modal ainsi qu'un attribut data-reveal.
La balise a ayant pour classe close-reveal-modal correspond au bouton de fermeture que l'on peut voir sur l'exemple.

On peut ainsi permettre à notre site d'afficher simplement des modals, que l'on peut entièrement personnaliser et qui s'adaptent bien aux différentes interfaces.

Intégration de contenu vidéo

Foundation permet également d'intégrer facilement du contenu vidéo dans vos pages web :

```
<div class="flex-video">
   <iframe src="http://player.vimeo.com/video/60122989" frameborder="0"
allowfullscreen></iframe>
</div>
```



Que ce soit dans une grille, un block, un modal ou autre, une vidéo, dans une balise iframe incluse dans un div ayant pour classe flex-video, va se redimensionner pour s'afficher correctement dans l'espace qui lui est attribuée.

Conclusion

Foundation est une technologie très pratique et facile à utiliser pour développer des applications web responsive, elle s'adapte facilement au format mobile et propose beaucoup de fonctionnalités pour personnaliser vos sites web.

REACT (> v0.12)

Introduction:

React est un framework Javascript développé par Facebook. L'objectif principal est d'optimiser les traitements de mise à jour du DOM (Objet Modèle de Document) pour que le rendu soit toujours le plus fidèle quelque soit les performances de la machine client.

La magie opère quand un serveur, pour lequel le langage d'exécution n'importe peu (Ruby, PHP, NodeJS, Python, etc …), hébergeant le code source redistribue les traitement entre lui et les clients. Pour ce tutoriel, nous utiliserons NodeJS.

Démarrage:

Pour la mise en place d'un modèle de projet assez simple, je vous invite à cloner le *repository* GitHub à cette adresse : https://github.com/koistya/react-static-boilerplate.git

C'est un petit espace de travail déjà préparé qui, une fois déployé à l'aide de la commande ***npm start*** (avoir installé NodeJS sur votre machine aupréalable), permet de développer avec une exécution en temps réel et un déboggeur.

Par défaut, l'application est déployée sur ***localhost:3000*** et un déboggueur React est disponible à l'adresse *localhost:3001*. Vous pouvez également installer le plugin React Developer Tools disponible pour Google Chrome si vous développez avec ce navigateur qui vous permettra d'observer le comportement des composants React.

React, Principes de bases :

Avec le *repository* que vous venez de cloner et de déployer, vous disposez d'un projet complet. Vous avez à votre disposition une page d'exemple ainsi qu'un menu basique contenant deux liens.

Ce starter est conçu pour que vous n'ayez pas besoin de vous prendre la tête avec les routes puisqu'il scanne le dossier pages et crée les routes en fonction de ces fichiers. Mais nous ne traîterons pas de cet aspect dans ce tutoriel.

Je vous invite à ouvrir le fichier pages/index.js afin d'en comprendre le contenu.

Notez qu'en utilisant ce boilerplate vous disposez de la syntaxe de classes ES6, vous n'êtes pas obligé de l'utiliser, mais je l'utiliserais dans ce tutoriel.

Bon, après cette introduction, étudions le fichier index.js que j'ai rédigé :

```
import React, {Component} from 'react';
import Grid from '../components/Grid';

export default class extends Component {
render() {
   return (
     <Grid items={12} onLayoutChange={this.onLayoutChange}></Grid>
   );
 }
}
```

Pour utiliser des composants de React, vous les importez en haut du fichier.
Toute classe React héritant de *Component* implémente une méthode de rendu (*render*) qui est appelée quand le composant est inclus en utilisant sont nom dans une balise XML comme c'est le cas pour le composant Grid.

Enfin, pour faire passer des paramètres et attributs entre les composants, on peut ajouter en paramètres de la balise. Ces attributs sont appelés props dans React.

Le composant Grid cité ici fait partie de mon application, c'est une autre classe Javascript React définie dans le dossier ../components/Grid/ :

```
import React, {Component} from 'react';
import ReactGridLayout, {Responsive} from
'react-grid-layout';
import _ from 'underscore';

import './Grid.scss';

export default class Grid extends Component
{
 static defaultProps = {
   className: "layout",
   rowHeight: 50,
   isDraggable: false,
   isResizable: false,
   cols: {lg: 4,md: 4,sm: 3,xs: 1,xxs: 1},
 };

 static propTypes = {
   onLayoutChange:
React.PropTypes.func.isRequired
 };

 constructor(props) {
   super(props);
   this.state = this.initialState();
   this.onBreakpointChange =
this.onBreakpointChange.bind(this);
 }

initialState() {
  return {
    layouts: {lg: this.generateLayout()},
    currentBreakpoint: 'lg'
  };
}

static generateContent(n) {
  var text = "";
  for (var i = 0; i < n*10; i++) {
    text += ""+n;
  }
  return text;
}

generateDOM() {
  return _.map(_.range(this.props.items),
function (i) {
    return (<div className="MaBoite"
key={i}>{Grid.generateContent(i)}</div>);
  });
}


render() {
  return (
    <div>
      <Responsive
```

```
generateLayout() {
   var p = this.props;
   return _.map(new Array(p.items),
function (item, i) {
     return {x: i%4, y: i, w: 1, h: 1, i:
i};
   });
 }

 onBreakpointChange(breakpoint) {
   this.setState({currentBreakpoint:
breakpoint});
 }

 onLayoutChange(layout) {
   this.props.onLayoutChange(layout);
 }
```

```
        layouts={this.state.layouts}
        onBreakpointChange=
            {this.onBreakpointChange}
        onLayoutChange=
            {this.onLayoutChange}
        useCSSTransforms={true}
        {...this.props}>
        {this.generateDOM()}
      </Responsive>
    </div>
  );
 }
}
```

Finalement, React permet de définir des classes similairement à tout langage de programmation orienté objet. Ici j'ai décidé de faire appel à un composant extérieur disponible dans le paquet react-grid-layout.

N.B: Si vous rencontrez des problèmes de dépendances, il faut supprimer les dossier des paquets de composants qui sont parfois inclus à l'intérieur de ceux ci pour que le module utilise ceux téléchargés dans leur plus récente version. React est en constante évolution puisque tout juste sorti de l'oeuf donc certaine dépendances en cache dans les paquets sont très vite désuettes.

defaultProps définie les propriétés par défaut du composants au cas où elle n'auraient pas été définies lors de l'appel, j'en parlais plus haut.

propTypes permet de définir des critères de dépendance à respecter, ici par exemple, si la méthode *onLayoutChange* n'a pas été définie dans l'appel du parent, une erreur est générée.

Enfin, le dernier mystère est l'attribut *state*. C'est dans celui-ci que sont stockées toutes les informations relatives à l'objet qui sont sujettes à modification. C'est cet attribut qui est observé en permanence et édité par des événements par exemple. Ce sont en quelques sortes toutes les métapropriétées de l'objet. Dans notre cas, la disposition des éléments de la grille (*layout*) est stockée dedans.

En l'état actuel de React, réaliser une application est très fastidieux puisqu'il faut définir tout de A à Z. Le coeur de React repose essentiellement sur le principe énoncé en introduction qui optimise les rendus clients à l'aide du serveur. Un langage qui permet de faire du responsive design, en partant de rien.

Tests des capacités d’adaptation

Pour tester l’adaptation des applications web réalisées nous avons utilisé l’outil de développement de google chrome permettant de tester les pages web sous différents formats.
Nous avons également réalisé quelques tests sur différents navigateur (firefox, IE).